AD-A043 484 LOCKHEED-CALIFORNIA CO BURBANK F/G 9/2
SUMMARY OF 1976 INDEPENDENT RESEARCH ON ENGINEER ORIENTED REMOT--ETC(U)
JUN 77 R W LINGARD

UNCLASSIFIED LR-28005 NL

1 OF 1
AD A043484

END
DATE FILMED
9-77
DDC

BURBANK, CALIFORNIA, U.S.A.

LOCKHEED · CALIFORNIA COMPANY
A DIVISION OF LOCKHEED AIRCRAFT CORPORATION

REPORT NO. LR-28005
DATE 6-30-77
MODEL Independent Research
COPY NO. 36

TITLE
SUMMARY OF 1976 INDEPENDENT RESEARCH ON ENGINEER ORIENTED REMOTE COMPUTING

REFERENCE 21-3715 5407
CONTRACT NUMBER(S)

PREPARED BY R. W. Lingard
R. W. Lingard, Sci. Comp. App. Spec., Sr.
Simulation Analysis Programming

APPROVED BY H P Weinberger
H. P. Weinberger, Group Engineer
Simulation Analysis Programming

APPROVED BY J D Little
J. D. Little, Department Manager
Scientific Analytical Programming

APPROVED BY R B Perry
R. B. Perry, Manager
Scientific Computing Division

REVISIONS

REV. NO.	DATE	REV. BY	PAGES AFFECTED	REMARKS

FORM 402-2

FOREWORD

This document is a report on the second year's accomplishments on the Calac independent research task entitled, "Engineer Oriented Remote Computing" (project number 76011102). The author is indebted to Dean Saiki who assisted in the design and implementation of many improvements to the production system. The author is also thankful for the many constructive suggestions made by Howard Weinberger and Thomas R. Jones during the course of this task.

ABSTRACT

The usefulness of computers in solving scientific problems is a function of the ease with which users can communicate with existing hardware and software. This research is aimed at improving such man-computer communication. Specifically, a computer system has been designed and partially implemented which will provide a software interface between users, possibly inexperienced in computer processing techniques, and available programs and analysis systems.

This system, denoted as ASSIST (A Scientific Software Interface System for Terminal-users), aids users in accessing and utilizing existing applications software from remote terminals. The system provides three basic functions. It helps users find programs relevant to their problems; it assists them in preparing required input data; and it aids in the actual submittal of programs and data for computer processing. In addition, the system monitors usage of the facilities and provides information for aiding management in making decisions for ensuring the efficient and proper use of available computer resources.

The basic approach has been to design a language which programmers can use to describe program characteristics (function, input format, submittal requirements, etc.). These descriptions can then be interactively interpreted by ASSIST to aid individuals wishing to use any available program. Thus, the user has a helpful interface system he can converse with while he is trying to find, prepare input for, or submit a program.

TABLE OF CONTENTS

INTRODUCTION

Technological advances in hardware have made computers practical and economical tools for ever increasing numbers of users. More and more people with less and less programming experience will be using computers in the years to come. No longer can systems be designed without consideration of these ultimate users. The effectiveness of future systems will be measured by the ease with which man can communicate with them.

Although a great quantity of problem solving software is available today, most is usable only by those possessing reasonable backgrounds in computing. In order to make most present systems and programs useful tools for the non computer oriented individual, they must either be redesigned, or some communication's interface between the user and the existing software must be developed. This latter approach, if general enough, could extend many existing computing capabilities to non programming users without costly redesign of applications software.

An experimental version of a system aimed at providing such an interface has been developed. This system, denoted ASSIST (A Scientific Software Interface System for Terminal-users), was designed to aid users in accessing and utilizing existing applications software from remote terminals. The system provides three basic functions for the user. It helps him find programs relevant to his problems; it assists him in preparing input for selected programs; and it aids in the actual submittal of programs and data for computer processing.

This report discusses the progress made during the second year (1976) of this task. The preliminary design of ASSIST and the first year's progress are contained in the report, "Engineer Oriented Remote Computing" (LR 27518).

SECTION 1.0

OVERVIEW

1.1 OBJECTIVE

The principal objective of this task is the development of a user oriented system to aid in the solving of problems on computers. This system, to be known as ASSIST (A Scientific Software Interface System for Terminal-users), will provide an interface between users, possibly inexperienced in computer processing techniques, and the existing hardware and applications software. The goals of this system are to reduce both job turnaround time and costs associated with the use of existing batch oriented software.

1.2 BACKGROUND

Over the years an extensive collection of application programs and systems has been produced. Making effective use of this software, however, usually requires both considerable knowledge of individual programs and familiarity with computer processing techniques in general. In particular, one must know what specific programs exist, what input data is required for each, in what form such data must be supplied, and what information must be provided to the computer operating system to effect the actual processing of programs and data.

Non programming users at Calac have traditionally depended upon professional programmers as their interface with existing software. In the computing environment prior to 1975 (Figure 1), it was the professional programmer who directly accessed both the computer and the library of existing programs. A non programming user, with a problem to solve, would

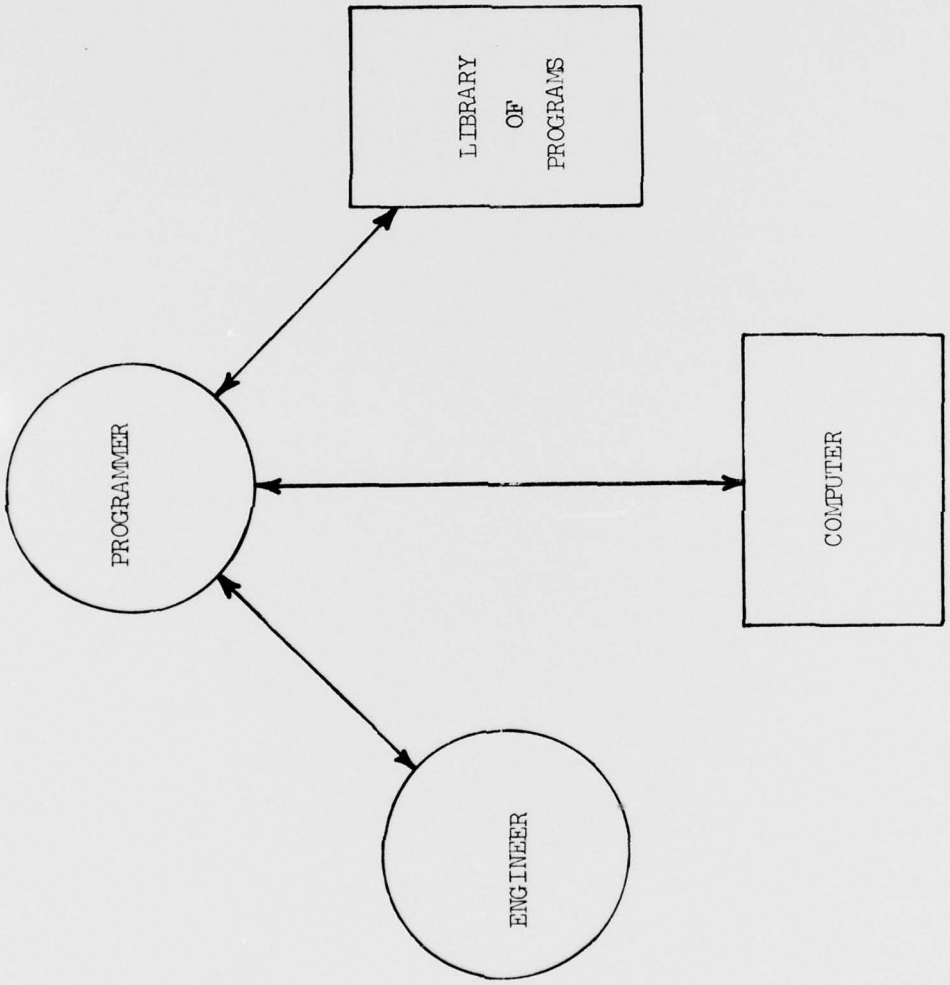

Figure 1. The Environment Prior to 1975

explain it to some programmer who would perform the necessary tasks to prepare a computer acceptable form of the problem (i.e., put together program and data with required system control information). The programmer would then accomplish the actual computer processing and return the results to the user. This mode of operation had obvious inefficiencies for many kinds of problem solving. There were often delays in processing and errors due to misunderstandings. With more and more people requiring more and more computer processing there was clearly a need to put them in closer contact with the computer.

As a consequence of this conclusion, a Direct Computer Access System, DCAS, was established which logically extended the computer to allow access to it through remote terminals (Figure 2). Initially DCAS existed only as a subset of IBM's Time Sharing Option (TSO). Although this gave a user direct access to the computer, it did not improve his access to the library of programs. This research deals primarily with the task of extending DCAS to improve the user's ability to directly utilize this collection of existing applications software. In effect, the goal is to automate the interface function previously provided by the programmer. In order to accomplish this the programmer must be provided with some means for transferring his "knowledge" (or information in his keeping) regarding the use of specific analysis software to DCAS. ASSIST is that augmentation of TSO which gives DCAS this capability.

1.3 SYSTEM DESCRIPTION

ASSIST has been designed to bring together information regarding existing application programs and potential users in an interactive environment (see Figure 3). For each program to be made available through ASSIST, certain descriptive information will be provided by a programmer. This information includes a general program description, a complete and and precise input specification, and certain job control information

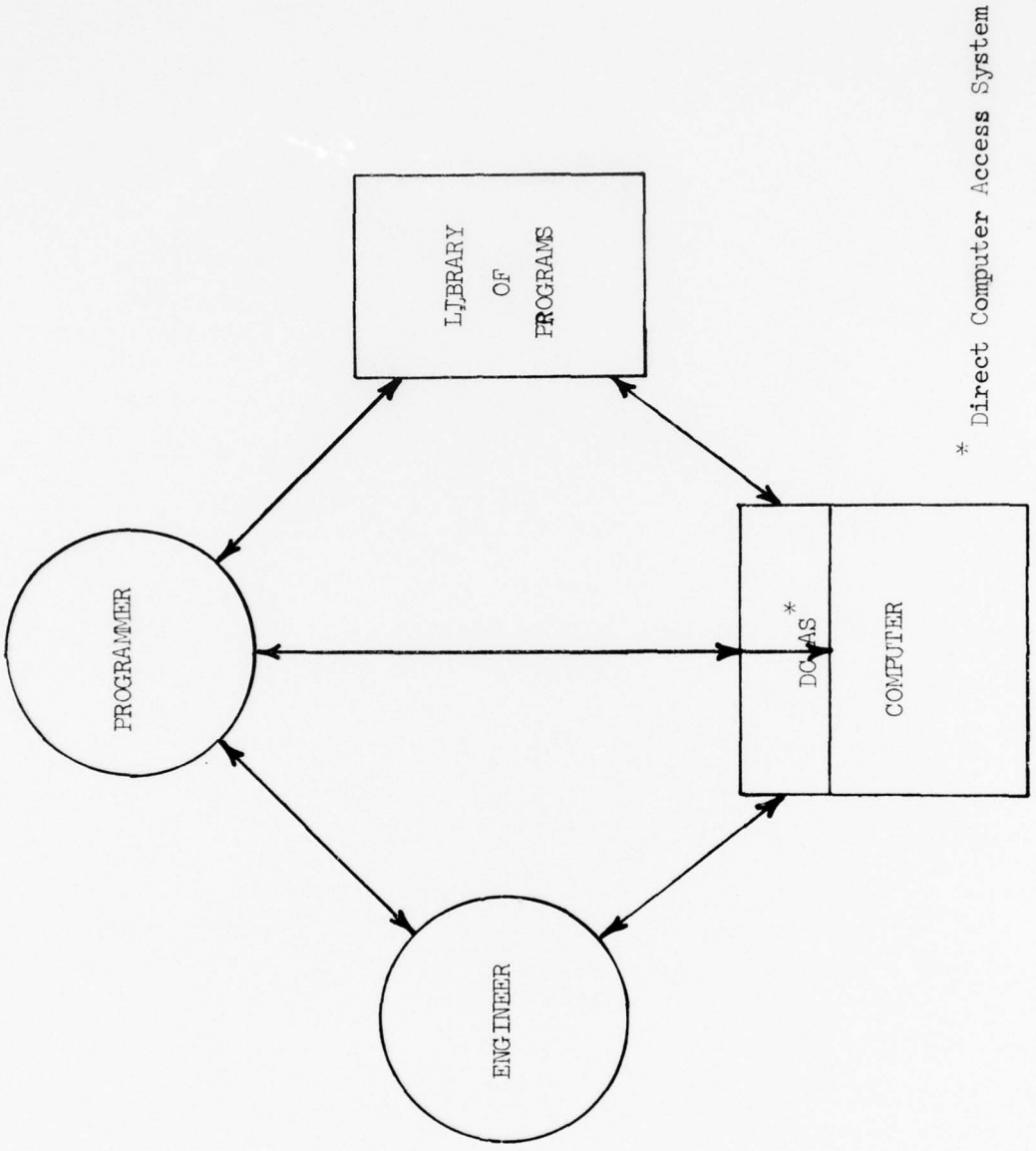

Figure 2. The DCAS Environment

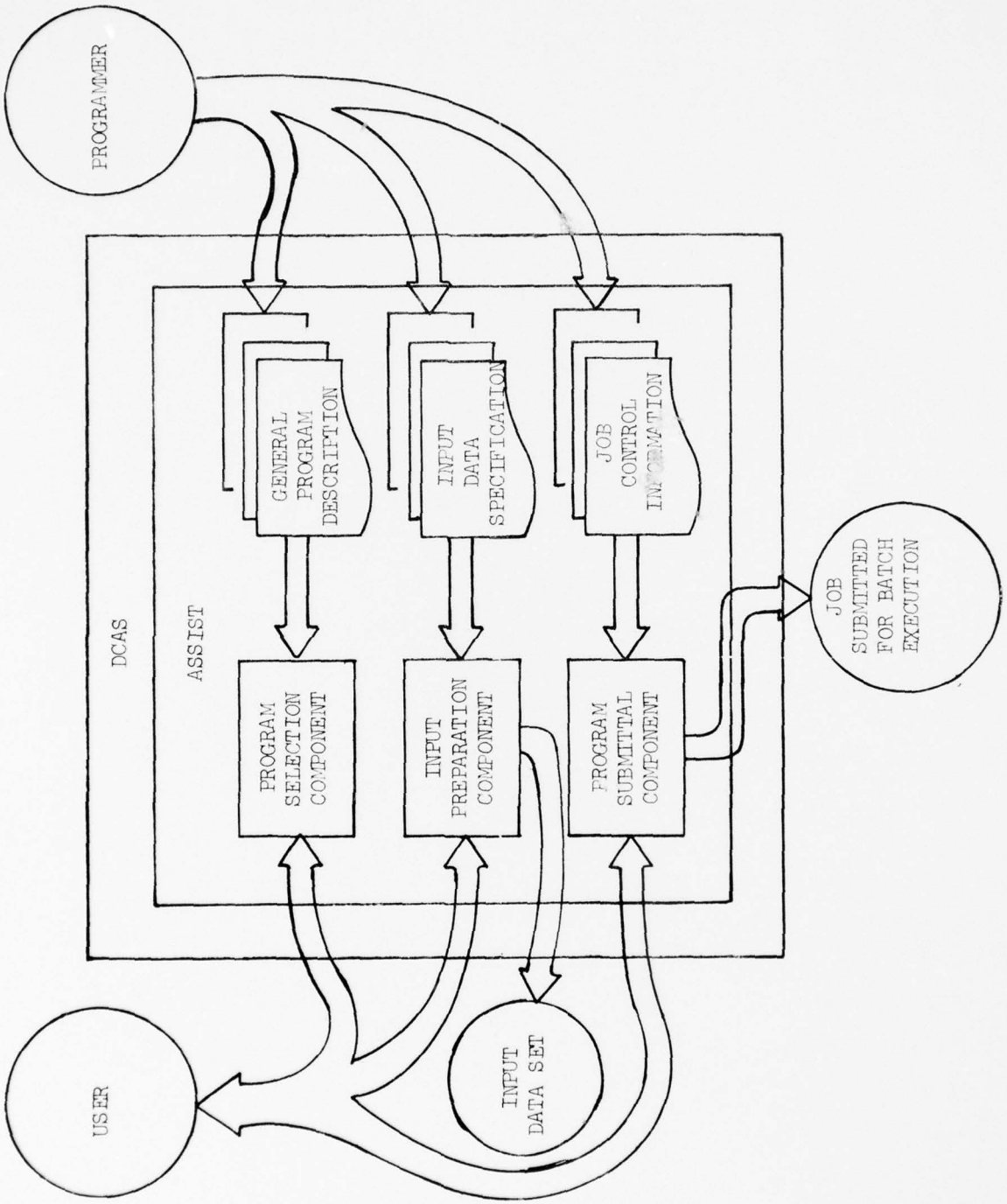

Figure 3. ASSIST

necessary for running the program on the computer. The user will then be able to interact with various components of ASSIST, which have access to this programmer supplied information, for solving some problem. If he needs information regarding the availability of certain software, he will access the Program Selection component. This will tell him about existing programs in a particular category he selects. Once he knows which program to run he may elect to access the Input Preparation component to assist him in preparing his data. He may ask to be prompted for every quantity needed or merely to have his data checked for completeness and, to some degree, correctness. Finally the user will be able to access the Program Submittal component which will automatically create all necessary job control information and submit the specified program and data for execution.

1.4 PROGRESS

During 1975 a preliminary design of ASSIST was completed. The component of the system which assists users in submitting programs was developed and put into controlled use for testing.

In 1976 an initial production version of ASSIST, containing extensive users aids for program submittal, was completed and put into use. Other aids were implemented and several more were designed including software to help users in preparing program input data. A detailed account of the 1976 progress on the various components of ASSIST is presented in the following sections.

SECTION 2.0

PROGRAM SUBMITTAL

2.1 OBJECTIVE

The purpose of this component of ASSIST is to simplify the task of program submittal by automatically generating the necessary job control information required by the operating system for program execution. The fundamental concept is that non programming users should not be required to learn the details of interfacing with the operating system in order to run jobs on the computer. The users should be able to communicate their needs in terms meaningful to them not in the language of the operating system. For example, a user desiring plot output should merely have to say, "PLOTS" or respond affirmatively to the question, "Do you want the output plotted?" rather than have to know how to appropriately modify the DD (Data Definition) statement of the associated plot file. Such capabilities could be of great benefit to the experienced programmer as well, for even with a knowledge of JCL (Job Control Language), it might be far simpler to allow ASSIST to automatically generate necessary control information. Certainly there is much less chance of error for either the experienced or inexperienced user when the program setup and control information are produced automatically.

2.2 APPROACH

The approach taken in this research for assisting users in program submittal has been to design a language, and interpreter for it, which can be used by programmers for expressing the information necessary for running programs on the computer. This language, known as the PSI (Program Setup Instructions) Macro language, is an augmented job control language (JCL) which allows construction of generalized sets of JCL for

the IBM operating system. The interpreter acts as a preprocessor or macro processor, expanding programs written in this language into complete and valid jobs to be executed on the computer.

In a typical case, a programmer who is familiar with a particular program and the JCL required to run it will develop a generalized set of job control instructions called a PSI macro. This PSI macro will then be placed in an on-line library and, hence, will be available to all users through the PSI macro processor known as RUNPROG. Once a PSI macro has been so created for a given program, users can run that program by accessing RUNPROG, without regard to any JCL concerns. Furthermore, changes that may be required in the JCL due to program modifications, system changes, or operational considerations can be usually made in the single version of the generalized JCL in the PSI macro library without requiring any change on the part of the users. In cases where programming changes were made, all users will automatically get access to the latest version of the program. Thus, this component of ASSIST can help not only the user in program submittal, but the programmer in program maintenance.

2.3 PROGRESS

During 1975 a preliminary version of the program submittal component (RUNPROG) was developed and put into limited production use. This version contained only capabilities for substitution of parameter values, obtaining user related information, simple communication with the user, and submittal of the complete job for batch processing. During 1976 the capabilities of the program submittal component were greatly expanded. The most significant of these are described below and a complete description of the current capabilities of RUNPROG is given in Appendix A, "Guide to Writing a PSI Macro."

2.3.1 Conditional Expansion

The first major capability added to RUNPROG was conditional expansion. A preprocessor statement similar to the IF statement of PL/I was implemented within the PSI macro language. With the "IF" a JCL statement or group of statements can be included or excluded in the job being built based upon some test. These IF statements can be nested to any depth.

2.3.2 Interactive Communication

Capabilities were implemented to allow communication with the user during execution of the PSI macro. A PUT statement allows the writer of a PSI macro to cause a message to be displayed at the user's terminal, and a GET statement permits the reading of a line of information from the terminal. Thus, a macro can be written which asks the user to supply some information and then reads in what is given.

2.3.3 Iteration

A capability, not in the original design of RUNPROG, was added which allows the repeated execution of a series of macro statements. The preprocessor statement used for this purpose is the "DO-WHILE" and is similar to the same construct in PL/I. The group of statements to be executed may include any number of JCL or preprocessor statements. These statements immediately follow the DO-WHILE statement and conclude with a preprocessor END statement. When execution reaches the END statement, control is transferred back to the corresponding DO-WHILE. Part of the DO-WHILE is a condition test just as in the IF statement. As long as the condition remains true the group of statements following the DO-WHILE is executed again. Once the condition fails, control is transferred to the statement following the associated END statement. DO-WHILE's can be nested to any depth.

2.3.4 Character Manipulation

A capability was added to the design and implemented which allows the extraction of a substring from a given string of characters. This function, called SUBSTR, operates identically with the corresponding PL/I function. Any consecutive string of characters can be extracted from a given string by specifying the beginning position and the number of characters to take.

2.3.5 Programmer Aids

Software has been designed and implemented to aid programmers in adding, modifying, and deleting PSI macros from the on-line library. A complete log of changes to PSI macros is maintained by the system. This log contains the time and date of the action, the identification of the programmer taking the action, the name of the macro involved, and the type of action taken. The software also checks the user's identification to ensure he is authorized to modify the macro library.

A capability has also been added to aid programmers in testing newly developed PSI macros. This feature, called TESTMAC, operates like RUNPROG except it accesses a macro in the programmer's library and directs a listing of the job built back to the terminal rather than submitting it to the computer for execution. Thus, the programmer can watch the job being built, card by card, and discover any errors as they occur.

2.3.6 Other Features under Development

The original design of RUNPROG included the capability to invoke one PSI macro from within another and the ability to copy a data set into the job being built. These capabilities are currently under development and will be implemented in the production version during the third quarter of 1977.

Also included in the original design was the ability to place arbitrary expressions wherever constants are allowed within preprocessor statements. Although this capability has been fully developed, the core limitations of the current TSO environment in which RUNPROG operates, have made its installation impossible.

The design of RUNPROG was modified to include as a diagnostic aid the capability to get a symbol table dump (i.e., a listing of all macro variables and their current values), and several new functions. In order to compensate for the lack of arithmetic capability of RUNPROG, INCR (increment) and DECR (decrement) functions were defined. INCR allows increasing the value of a quantity by some given amount while DECR allows for reduction in a similar manner. Three other functions added to the design were INDEX, LENGTH and CONCAT. These are string manipulation functions which correspond exactly with similar constructs in PL/I. INDEX allows the determination of whether one string is a substring of another and, if so, returns the position at which the substring begins. LENGTH returns the number of characters in a given string, and CONCAT allows the concatenation of two strings. These functions are scheduled for implementation during the second quarter of 1977.

Other extensions of RUNPROG which will be studied in 1977 include the GO-TO statement, a dynamic naming or array facility, and the ability to do indirect addressing. Also during 1977, effort will be directed at improving the responsiveness of RUNPROG.

SECTION 3.0

INPUT PREPARATION

3.1 OBJECTIVE

The purpose of this component of ASSIST is to aid the user in preparing input data for a program he has chosen to run. It will do three things for the user. It will tell him what input quantities are required for a given program; it will enable him to provide those values in a convenient manner (without requiring that he know the specific data formats required by the program); and it will check the data he provides both for completeness and correctness. Furthermore, this component can be used interactively, prompting the user when necessary and allowing him to correct errors as they are discovered. The effect of this component is to put a user's guide to a program on-line and in such a way that the user can converse with it.

3.2 APPROACH

The approach taken for this component is basically the same as that used for program submittal (i.e., RUNPROG). In particular a language has been developed which allows programmers to describe program input requirements. Actually this language is just an extension of the PSI macro language since the fundamental requirements of this component are identical with those of program submittal. Namely, it must interact with a user, providing some information and obtaining other information, and based on that, construct a data set. In the case of program submittal, the data set built is job stream input while in the case of input preparation, it is a data set for input to some program. These differences in no way effect the logical operation of the component of ASSIST which interprets programmer written descriptions and interacts with users.

All that is required, therefore, to be able to provide assistance to the user in input generation is to add certain constructs to the existing macro language. In particular, constructs are needed which allow the description of required input parameters, including types and ranges of acceptable values, and the format in which the program expects them to be given.

With such an expanded macro language, a programmer can describe the input requirements for any program in such a way that data prepared for that program can be automatically checked for completeness and correctness, and, if desired, the input can be prepared in an interactive mode. In the latter case, information will be requested of the user through an input description (ID) macro written by a programmer. Normally, this request will be a list of input parameters for which the user must supply values. Additional messages can be displayed to the user at the discretion of the programmer writing the ID macro. The user can ask for a description of any parameter requested and the values he supplies will be checked for proper type (e.g., character, integer, etc.) and limits (e.g., $0<X<10$) according to information specified in the ID macro. The user will be immediately notified of any errors and allowed to correct them. The input values will then be formatted as required by the program. When this component is used just to check a prepared data set, the data set must be properly formatted. In this case, a list of all discovered errors will be returned to the user. The use of this component could result in significant savings of computer resources by helping users to prepare program input data which are correct the first time.

3.3 PROGRESS

A software specification for the extended macro language has been developed, and the design, coding and testing of the required software modules are in progress. Implementation will be accomplished during the

third quarter of 1977. The major features added for the purpose of describing program input requirements are given below. A complete description of the language constructs for the ID language is contained in Appendix B.

3.3.1 Parameter Description

A capability has been included to allow a description to be associated with each input parameter. This is accomplished with a DEFINE statement. When a variable is so defined, its description is available to users on request during input preparation by typing a "?" and the name of the parameter or parameters desired.

3.3.2 Format Description

A statement is included for describing input formats. It can be used to specify a list of parameters and the format in which they must be supplied to the program. It is an extension to the PUT statement and is identical in form to the PUT EDIT capability in PL/I. However, instead of taking elements in the list, converting them, if necessary, and outputting them according to the format given, the PUT statement takes the elements and checks them against any type or limit specifications given in the macro (see below) and then, only if they are correct, outputs them as specified in the format. Errors in input values will be displayed to the user and he will be allowed to correct them. The PUT statement will then be re-executed automatically.

3.3.3 Type Specification

A TYPE statement has been included which allows specification of the type requirements, if any, which a parameter or set of parameters must satisfy. For example, a set of parameters can be required to be

integers or real numbers or characters. The PUT statement will check each parameter in the list to see if a type specification has been given for it. If so the value supplied will be checked against the type specified.

3.3.4 Limit Specification

In a similar manner a range or set of acceptable values can be specified for sets of parameters. This is done with the LIMIT statement. For example, a parameter can be required to be between the values 0 and 100 or be one of the values 0, 1, or A. Limit specifications are handled in exactly the same manner as type specifications.

SECTION 4.0

PROGRAM SELECTION

4.1 OBJECTIVE

The purpose of this component of ASSIST is to provide information to the user regarding available applications software. The nature of this information will be such that he may determine which, if any, available programs might be applicable to a given problem. Program titles, abstracts, development and revision dates, names of responsible programmers, and program identification (Reference File) numbers are examples of the information to be provided. Additionally, this information will be made conveniently accessible from a terminal. Specifically, a user will be able to give a keyword or list of keywords and a list of program titles will be searched for the occurrence of the word or words given. The program titles corresponding to matched keywords are returned to the user. He may then list the abstract and other information desired for selected programs. One of the primary advantages of such a capability is that it provides an effective means for disseminating information regarding available software throughout a large community of users.

4.2 APPROACH

The approach taken in the development of this component has been to create an on-line data set containing an entry for each PSI macro accessible through the program submittal component. In some cases there may be more than one macro for a given available program, but there is always at least one. The entry for a given macro contains its name (a one to eight character identification), a title, the name of the responsible programmer, and

the program reference file (RF) number of the program accessed by the macro. This RF number is a key into an existing data base of program description information, the Program Reference File. This data base is maintained by the Scientific Computing Division and contains the remainder of pertinent information for applications programs. This data base will be placed on-line and the capability to access it from terminals will be provided.

4.3 PROGRESS

An on-line data base containing entries for all PSI macros was created, and software was developed to permit accessing this information from terminals. In addition, facilities were developed for adding, deleting, and modifying the data base information. In fact, the software which allows programmers to add PSI macros was designed to require that a descriptive title be given before the macro is added to the macro library. When a macro is modified the descriptive title can be changed and when a macro is deleted so is its title entry. Thus, the data base of descriptive titles always reflects the current status of available programs. Two commands have been provided for accessing information in this data base. The first command, called DESCMAC, can be used to obtain the descriptive title for a PSI macro by giving its name. The second, SCANMACS, scans the entire data base for a keyword supplied and returns all entries containing that word.

Although the Program Reference File data base has been placed on-line, the link between it and the PSI macro data base has not been completely established, and the capability to access it from a terminal has not been developed. These items are scheduled for completion during the third quarter of 1977. Also the feasibility of extending the searching capability to include Boolean combinations of keywords will be studied. This capability would, for example, allow a user to ask for all macro titles containing both the words "structural" and "analysis" or just the

single word "structures" or to ask for all titles containing the word "loads" but not the word "dynamic". A second extension will also be studied which would allow direct searching of the Program Reference File data base, including abstracts, in a similar manner. These extensions will be evaluated during the fourth quarter of 1977.

SECTION 5.0

USAGE MONITORING AND CONTROL

5.1 OBJECTIVE

The purpose of this component of ASSIST is to ensure the efficient usage of available computer resources by developing adequate system controls and through the monitoring of user activity. Since extensive computer software and hardware resources have been made available to non programming users, there is a need to prevent inadvertent misuse due to lack of computer experience. As a minimum, sufficient information must be collected in order to determine whether the resources are being efficiently utilized. The information so collected, since it will reflect user activity, will also be valuable for guiding efforts to improve the efficiency of ASSIST itself.

5.2 APPROACH

In an environment where virtually all computing capabilities have been given to non programming users, it is a practical impossibility to protect the hardware and software from deliberate misuse. The approach in this research is twofold. Firstly, wherever possible, the accidental misuse of resources will be prevented. All components of ASSIST have been designed so that the checking of user supplied values and requests is possible, and in many cases the required parameters which effect the usage of resources are automatically determined. Secondly, statistics regarding the usage of resources will be collected. While misuse, either accidental or deliberate, can not be prevented this way, it can at least be discovered after the fact. This can, of course, be a great deterrent to potential misusers as well as a means to correct practices which result in inefficiencies.

5.3 PROGRESS

The accomplishments toward prevention of accidental misuse of resources were designed and implemented within the Program Submittal and Input Preparation components of ASSIST. By their very nature these components eliminate many sources of user errors. The Program Submittal component automatically determines many required parameters, and both components have capabilities for checking the correctness of user supplied values. In the case of program submittal, the computer resources requested (e.g. core, time, lines of output, etc.) can be controlled, and, in the case of input preparation, the execution of runs with erroneous data can be prevented.

Beyond these capabilities, the primary method for ensuring the efficient use of resources has been through the collection and analysis of data relating to user activity. Much of the relevant data is available for terminal sessions just as it is for normal batch work through the standard accounting procedures. In addition to this data, a module has been designed and implemented experimentally which collects information regarding batch submittals made by terminal users. The standard accounting procedures, of course, collect the same information about these remotely submitted runs as they do for normal batch jobs. This new module, however, additionally collects the following information for each run.

- Userid of submitter
- PSI macro used
- Time, line, and card estimates
- Priority requested
- Date and time of submittal

This data can be used not only to ensure the proper use of resources by terminal users, but to determine the level of use and actual

users of each PSI macro. Although this module has been coded and tested, it has not been implemented in a production mode because the current response problems during submittal will not permit the monitoring of activity. Once response improves to a satisfactory level, this module can be added.

During 1977 the feasibility of monitoring other activities of users will be considered and software will be developed to produce meaningful reports from all available information on user activity.

SECTION 6.0

CONCLUSIONS

The success experienced with early versions of ASSIST has continued with improved implementations. The present version clearly demonstrates the practicality of augmenting existing application software with a communications interface system. Some of the specific benefits of this system are as follows:

(1) By eliminating the need for communication with professional programmers, errors due to misunderstandings disappear and elapsed processing time is reduced.

(2) By providing simple and direct communication with the computer, the productivity of the user is increased.

(3) By checking user input data and automatically generating job control information, execution errors are greatly reduced.

(4) Making information about software centrally and easily available and providing a convenient means for using it results in a greater utilization of available software and less duplication of program development of effort.

By the end of 1976, 180 programs, or versions of programs, were directly available to users through ASSIST. Included in this number are several major computing systems, such as FAMAS (Flutter And Matrix Algebra System) and the NASTRAN (NAsa STRuctural ANalysis) system. In addition, ASSIST includes utility functions to allow users to print and punch data sets and to compile small FORTRAN and PL/I programs. The number of users

at the end of 1976 was 172, and more than 100 runs were being made daily through ASSIST. ASSIST has been used to great advantage in almost every area of engineering analysis, as well as some areas of financial analysis, and recently in manufacturing. Significant cost savings have resulted, primarily due to reductions in elapsed time to complete analyses and through the elimination of many sources of error. For example, the structural engineering organization has reported that ASSIST was, in part, responsible for obtaining a dramatic improvement in both cost and schedule in determining external strength level loads for the L-1011-500.

Future developments in the area of user oriented remote computing will be at two levels. For the short term, efforts will be directed at improving the performance characteristics and basic capabilities of the present version of ASSIST. The major effort, however, will be directed toward determining the proper hardware and software configuration for providing users with remote computing capabilities. Both the short term and long term efforts have been strongly influenced by the results of a questionnaire which was distributed to DCAS users on May 20, 1976. The complete results and analysis of this questionnaire are given in Appendix C.

Although the responses to the questionnaire indicated a highly favorable attitude toward ASSIST, there were many constructive suggestions for improvement. The main criticism of the system was its poor responsiveness, a problem not with the ASSIST software but with the overall hardware and system software environment in which ASSIST is imbedded. Neither the IBM 360/91 hardware nor the OS/MVT software (the current operating system) were designed to support time sharing applications. Consequently, TSO, IBM's time sharing system which hosts ASSIST, has severe performance problems. Furthermore, due to the wide acceptance and overall success of DCAS, the number of users is expected to grow steadily over the next several years, adding to the difficulty of providing adequate performance.

Several approachs toward improving system responsiveness and capacity are possible. The first is simply to dedicate sufficient computing resources to achieve the desired level of response for the current number of users, an expensive but simple solution. The second approach is to replace TSO with a time sharing system designed to compensate for some of the inadequacies of the existing hardware and operating system software. Although such systems exist, none offers the range of capabilities available under TSO, and extensive retraining of users would be required if such a change were made. Because systems still must contend with hardware and operating system software which are inherently inefficient for the purposes of time sharing, a truly efficient replacement system is impossible. A third approach is to move the interactive functions from the IBM 360/91 onto a machine or machines better designed to support such activity. In this environment, a user would be connected to a satellite computer capable of handling interactive functions directly and be connected to the IBM host machine to provide remote batch capabilities. Such a distributed system, while possibly solving the major performance problems, introduces many other potential difficulties. Not the least of which are those related to the communication between the satellite computer(s) and the host. The major effort on this research task during 1977 will be to determine the potential value and practicality of such a distributed system for providing user-oriented remote computing capabilities.

APPENDIX A

GUIDE TO WRITING A PSI MACRO

A.1 INTRODUCTION

A PSI (Program Setup Instructions) macro is just a generalized set of JCL (where JCL is taken to mean all actual JCL, LASP control statements, linkage editor input, etc.). This generalized JCL is analyzed by a preprocessor (RUNPROG), and from it a complete job for batch execution is built. In writing a PSI macro it is usually best to start with a set of JCL as would be required for a batch submittal and make modifications to it as indicated in the following sections.

A.2 STRUCTURE OF A PSI MACRO

A PSI Macro can contain three types of statements:

(1) System Control Statements

These are the IBM System/360 JCL statements, LASP statements, linkage editor input, and other data. These obey the normal rules for syntax except they may optionally contain PSI variables (see section A.4).

(2) Comment Statements

Comments may be written in columns 2-80 of a card if a 'C' is placed in column 1. Comment cards are ignored by the preprocessor.

(3) Preprocessor Statements

These are special statements which control the creation of the job to be submitted. These statements are coded in columns 2-72 and

must be prefixed with a '%' symbol in column 1. These statements are stream rather than card oriented and hence each must be terminated with a semicolon. The preprocessor statements currently available are:

(a) The Macro Definition Statement (See Section A.6)
(b) The Macro End Statement (See Section A.6)
(c) The Input Statement (GET - see Section A.9)
(d) The Output Statement (PUT - see Section A.9)
(e) The Conditional Statement (IF - see Section A.10)
(f) The Group Delimiting Statements (DO & END - see Section A.10)
(g) The Assignment Statement (See Section A.11)
(h) The Iteration Statement (DO WHILE - see Section A.12)

A PSI macro should begin with a macro definition statement and end with a macro end statement (although neither is presently required). The other preprocessor, comment, and system control statements can occur anywhere within a PSI macro.

A.3 NOTATIONAL CONVENTIONS

When describing the syntax of preprocessor statements, the following notational conventions will be used.

(1) The brackets, '[' and ']' will surround items which may optionally be present.

(2) The braces, '{' and '}' will mean that one of the enclosed items must be chosen.

(3) Strings of lower case letters are used to indicate variables which must be replaced with some value.

(4) Strings of upper case letters indicate information which must be present exactly as written.

(5) The ellipsis, '...', indicates that the previous item may be repeated an arbitrary number of times.

For example, in the specification

ASSIGN identifier [{+ -} constant] ... TO subscript

'ASSIGN' and 'TO' must appear literally, 'identifier', 'constant', and 'subscript' are variables which must be replaced with values whenever they occur, '{+ -} constant' is optional and it may occur sequentially any number of times, but each time a choice must be made between '+' and '-'.

A.4 PSI VARIABLES

Any item (or portion of a statement) within a JCL set which may assume different values with different submittals may be given a symbolic name so a value can be assigned when the actual submittal is made. This name can consist of up to eight (8) characters. It must begin with a letter and can contain only letters, digits, break characters (_), and national characters ($,#,@). This symbolic name can be used for any item in a JCL statement by enclosing it between the symbols '<' and '>' and writing this string in place of the given item. For example, suppose the running time for some program varies with the input data. One could then write the following statement:

```
/*MAIN LINES=20,CARDS=50,TIME=<RUNTIME>
```

This would allow specification of value for the variable RUNTIME when the program is submitted for execution.

When the substitution of a value occurs, that value normally replaces exactly the string '<' followed by the variable name followed by '>' (for exceptions to this see Section A.5). If the length of the value is less than that of the string it replaces then the remainder of the statement will be moved left so as to immediately follow the value inserted. Correspondingly if the value is greater in length than the string it replaces, the remainder of the statement will be shifted right an appropriate number of spaces. If more than one variable is to be replaced in a statement, replacement proceeds from left to right.

This substitution of values can cause a statement to exceed the maximum allowable length for that type of statement. For actual JCL and LASP statements, RUNPROG automatically produces continuation cards when this condition occurs. All other statements are truncated after 72 columns.

PSI variables may also be used in preprocessor statements as in any programming language. In this case they are not surrounded by the delimiters '<' and '>'. The same variable name may appear in both a system control statement (surrounded by '<' and '>') and in a preprocessor statement and will have the same value in both contexts.

A.5 SPECIFYING COLUMNS FOR STRING REPLACEMENTS

Sometimes it is necessary to insure that the value replacing some variable name begins in a particular column. This can be accomplished by prefixing the variable name with the symbol '@' and the desired column number. For example, an accounting card could be written as follows:

```
/*AC<USERNAME><@19$CSR#><@33CODE><@34PROG#><@44$DP#>...
```

In this case the value of $CSR# would begin in column 19, the value of CODE in column 33, and so on. Replacement proceeds from left to right so if the same columns are affected by two replacements the latter one overrides the former. For example, if the value of CODE had length 2 it would occupy columns 33 and 34 after the replacement. However, the replacement of PROG# by its value would modify column 34 again and the second character of CODE in this statement would be destroyed.

A literal string can also be placed in a specified column by enclosing it within quotes. For example, the accounting card could be rewritten as follows:

```
/*AC<USERNAME><@19$CSR#><@33'1'><@34'1234'><@44$DP#>...
```

This would place a '1' in column 33 and the string '1234' in columns 34 through 37.

A.6 NAMING A PSI MACRO

The name of a PSI macro should begin with the letter 'P', which should be followed by the reference file number for the program being executed, and this can optionally be followed by any string of alphameric characters. The total length of the name, however, may not exceed 8 characters. Other names may be used if there is sufficient reason and if prior approval is obtained.

This macro name should appear on a macro definition statement which should be the first statement of the macro. The form of this statement is as follows:

```
% name: MACRO  identifier=constant [,identifier=constant] ...;
```

where 'name' is the PSI macro name
'identifier' is any variable name
'constant' is an integer or character string default value for the variable (see Section A.7)

The macro name may also appear on the macro end statement which should be the last statement of the macro. The form of this statement is as follows:

% MEND name ;

where 'name' is the PSI macro name.

As an example suppose a PSI macro were being developed to execute a program whose reference file number was '1234'. The macro would probably be named P1234 and would begin with the statement

% P1234: MACRO;

and end with

% MEND P1234;

This macro name will also be used in storing the macro into the PSI macro library (see Section A.14).

A.7 ESTABLISHING DEFAULT VALUES FOR VARIABLES

Default values may be given to PSI variables by listing them on a macro definition statement (see Section A.6 for a complete description of this statement). For example, suppose one wanted to have a default running time of 2 minutes for some program but still allow the user to override the time if necessary. This could be accomplished by using the following macro definition statement.

% P1234: MACRO RUNTIME=2;

Default values can be established only for those variables which have not been assigned values prior to invocation of the macro. That is, some

variable names have built-in default values, and the macro definition statement cannot be used to specify different default values. A complete list of these system variables and their defaults is given in the Table on the following page.

The system variables fall into 2 classes.

(1) User Information Variables

These contain information relating to the user and their values are automatically determined based upon the "userid" and "account" specified by the user at logon.

Specifically they are:

(a) $USERID - userid
(b) $CSR# - dcas account number
(c) $CHARGE# - class-work order, ewa, and serial number associated with dcas account number
(d) $DP# - department number associated with account number
(e) $GP# - group number associated with account number
(f) JOBNAME - name to be given the submitted job
(g) USERNAME - user name associated with userid
(h) USERDEPT - department number associated with userid
(i) BIN# - address (i.e. building number or bin number) where output is to be sent

(2) Convenience Variables

The remainder of the variables listed in the Table were provided merely as a convenience in using the early versions of RUNPROG which did not allow the setting of default values. At some time in the future the default values for all of these variables will be eliminated. Therefore, new macros should explicitly specify the defaults for any of these variables used, and existing macros should be so modified.

Variable Name	Default Value
$CARDS	'0'
$CLASS	'B'
$CHARGE#	automatically determined
$CSR#	automatically determined
$DP#	automatically determined
$GP#	automatically determined
$LINES	'5'
$PRTY	'0'
$REGION	'200K'
$TIME	'1'
$USERID	automatically determined
BIN#	automatically determined
CODE	'2'
DATA	a null data-set
JOBNAME	automatically determined
OBJECT	a null data-set
PROG#	'469000'
PRTY	'0'
SOURCE	a null data-set
USERDEPT	automatically determined
USERNAME	automatically determined
XCHAR	'A'

Table. RUNPROG System Variables and Defaults

A default value may be any preprocessor constant. Preprocessor constants may be numbers or character strings. A number may be written as a fixed point or floating point decimal number. A fixed point decimal number consists of 1 to 15 decimal digits with an optional decimal point. If no decimal point appears, the point is assumed to be immediately to the right of the rightmost digit. A floating point decimal number is written as a fixed point decimal number followed by the letter E, followed by an optionally signed decimal integer exponent (of no more than 2 digits). Any numeric constant may optionally be preceded by a plus or minus sign. Blanks may not appear within a numeric constant. The following are examples of valid numeric constants.

```
3.141593
732
003
.0012
3141593E-6
7.32E+2
.003E3
```

A character string constant is any string of up to 80 valid characters enclosed within single quotation marks. If a single quotation mark is a character in the string, it must be written as two single quotation marks with no intervening blanks. If two single quotation marks are used within the string to represent a single quotation mark they are counted as a single character. A null character string constant is written as two quotation marks with no interleaving blank. Examples of character string constants are:

```
'TITLE'
'SHAKESPEARE''S''''HAMLET'''''
'3.141593'
'' (null character string constant)
```

A.8 OVERRIDING DEFAULT VALUES

If a default value has been provided for a variable, it may be overriden by the user as follows:

(1) He must include the specification "MOREPARM(YES)" when using the RUNPROG command (See Section A.16). For example:

```
RUNPROG P1234 DATA(X.DATA) MOREPARM(YES)
```

(2) RUNPROG will then give him the opportunity to override default values. For each variable he wishes to change he must type the variable name followed by an equal sign followed by the value he wishes that variable to have (character string values must be enclosed in quotes but numeric values need not be). Multiple specifications can be given by separating them by blanks or commas. Entering a null line terminates this mode and RUNPROG resumes processing of the macro.

A.9 COMMUNICATING WITH THE USER

Input (GET) and output (PUT) statements are available for obtaining and providing user information. The GET statement has the form,

```
% GET identifier;
```

where 'identifier' is any variable name

The execution of such a statement will cause one line (80 characters) of information to be read from the user's terminal. Trailing blanks are eliminated and the resulting string is stored as a value for the variable specified by 'identifier'. A null line or a line of blanks are interpreted as a single blank character. A separate GET statement is (presently) required for each value to be obtained.

Since execution of the GET statement will cause the program to pause and wait for the user to respond, it is usually necessary to give him some prior indication as to what information will be required of him. This can be done with the PUT statement. In fact the PUT statement can be used anytime it is desired to issue a message to the user. It's form is as follows:

% PUT {identifier / constant};

where 'identifier' is any variable name

and 'constant' is any preprocessor constant (See Section A.7)

The execution of a PUT statement will cause the value of the identifier or constant specified to be displayed at the terminal. A separate PUT statement is (presently) required for each value to be output. An example of the use of the GET and PUT statements is as follows:

```
% PUT 'PLEASE SUPPLY A VALUE FOR X';
% GET X;
```

Communication with the user can also occur without the explicit use of the GET and PUT statements. Whenever a variable is encountered for which no value has been assigned (either by the system or from some prior statement within the macro), RUNPROG will prompt the user for a value. For example, if no value had been given for the variable RUNTIME, the processing of the statement.

```
/*MAIN LINES=20,CARDS=5,TIME=<RUNTIME>
```

would result in the following message being displayed at the user's terminal.

```
RUNTIME = ?
```

RUNPROG would then accept a value from the user for this variable, just as if a GET statement had been executed, and process the statement using the supplied value. This variable will now have the value specified by

the user, unless altered by an assignment statement (see Section A.11) or an explicit input statement, for the remainder of the processing of the macro. Hence, if another statement containing the same variable name is encountered,

```
// EXEC FORTLG,PARM.G=<RUNTIME>
```

the same value will be used and the user will not be prompted again.

The user may also specify values for variables using the same technique as was required for overriding default values (see Section A.8) even for variables with no established defaults. The user could thus supply all required inputs at once and avoid being prompted for each item individually except in cases where the explicit input (GET) and output (PUT) statements are used. (The GET statement will always require the user to supply a new value for a variable even if the variable already has a value).

A.10 SPECIFYING ALTERNATIVES

PSI macros may be written in such a way as to cause some JCL statements to be included in the job being built only under certain conditions. This may be accomplished by use of the preprocessor conditional statement. For example, suppose a given program could optionally produce plot output. The macro could be written in such a way as to include the setup and DD cards for the plot tape and the necessary instructions to the operator (via operator cards) only for the runs which actually generate plots. A variable, say PLOTS, could be selected as the test variable and one could write

```
% IF PLOTS='YES' THEN
/*SETUP DDNAME=PLOTTAPE,...
```

This would cause the setup card to be included only if the value of PLOTS was 'YES'. If PLOTS had not been assigned a value prior to execution of

the IF statement then the user would be prompted for a value. If more than one statement is to be optionally included based upon some test then that group of statements must be preceded by the preprocessor statement,

```
% DO;
```

and followed by the preprocessor statement

```
% END;
```

For example, one might write the following:

```
% IF PLOTS='YES' THEN
% DO;
/*SETUP DDNAME=PLOTTAPE,...
/*OPERATOR...
/*OPERATOR...
% END;
```

In this case the setup and operator cards will all be included or all omitted depending upon the value of PLOTS.

The conditional statement must have the following form.

% IF { identifier / constant } relop { identifier / constant } THEN statement-block

where 'identifier' can be any variable name

'constant' can be any preprocessor constant (see Section A.7)

'relop' must be either the relational operator '=' (equal) or '¬=' (not equal)

'statement-block' can be either a single statement or a group of statements preceded by a DO statement and followed by an END statement.

Conditional statements may be nested to any depth. The following are valid examples of the use of IF statements.

```
% IF A=1 THEN
%   DO;
      (group of statements)
```

```
%     IF B=2 THEN
%        DO;
            (group of statements)
%           IF C=3 THEN
               (statement)
%        END;
%  END;

% IF A=1 THEN
%   DO;
      (group of statements)
%     IF B=2 THEN
%        DO;
            (group of statements)
%        END;
      (group of statements)
%  END;
```

A.11 ASSIGNING VALUES TO VARIABLES

A limited form of an assignment statement is available in the present version of RUNPROG. This statement is written as follows.

```
% identifier = constant;
```

where 'identifier' is any variable name

'constant' is any preprocessor constant

One example of the use of this statement is the following:

```
% PRTY=3;
% IF TIME 7=2 THEN
% PRTY=1;
```

Here PRTY is given the value 3 as long as TIME is 2 (presumably the default value), otherwise PRTY is changed to 1.

A.12 PERFORMING ITERATION

Sometimes it is necessary to repeat a statement or group of statements within a job being built. This repetition can be accomplished with the preprocessor DO WHILE statement. It's syntax is as follows:

DO WHILE ({ identifier / constant } relop { identifier / constant });

where 'identifier' is any variable name.
'constant' is any valid preprocessor constant,
'relop' is either the relational operator
'=' (equal) or '¬=' (not equal)

The end of the repetition loop is indicated by the preprocessor END statement. The loop may contain statements of any type (preprocessor, JCL, etc.), and the group will be repeated as long as the condition of the DO WHILE remains true. For example, suppose the input data to some program could exist on several data-sets which would have to be concatenated together in order to make a run. Rather than requiring the user to do the necessary merging, the following code could be included within a PSI macro.

```
//SYSIN DD DSN=<INPUT>,DISP=SHR
% PUT 'SUPPLY NEXT INPUT DATA-SET NAME';
% PUT 'OR HIT CARRIER RETURN IF NONE';
% GET INPUT;
% DO WHILE (INPUT ¬= '');
// DD DSN=<INPUT>,DISP=SHR
% PUT 'SUPPLY NEXT INPUT DATA-SET NAME';
% PUT 'OR HIT CARRIER RETURN IF NONE';
% GET INPUT;
% END;
```

After the initial 'DD' card is written into the job stream the user is prompted for another data-set name by use of the PUT and GET statements (see Section A.9). The loop is then entered provided the value supplied was not null. A concatenation 'DD' card is generated and the user is again prompted for another data-set name. Control returns to the DO WHILE statement and the loop is exeeuted again as long as the last value provided was not null. This process continues until a null value is given for a data-set name. Control then transfers to the statement following the END statement.

A.13 BUILT-IN FUNCTIONS

The built-in functions which are available for use in writing PSI macros are described below.

A.13.1 DSN Built-in Function

Definition: DSN converts a valid TSO specification of a data set name into a form acceptable for use in a Data Definition (DD) statement, returning the converted form to the point of invocation. (Within TSO a user may refer to one of his data sets without explicitly including the top level index, namely his "userid". He may also refer to his own or any other catalogued data set by specifying the full data set name and enclosing it within single quote marks ('). In a DD statement, however, the full data set name, without enclosing quotes must be given).

Reference: DSN (string)

Argument: The argument "string" represents the string from which a full data set name is to be constructed.

Result: If "string" has enclosing quotes the value returned by this function is the "string" with the enclosing quotes removed. Otherwise, the value returned is a string consisting of the current user's "userid" followed by a point (".") followed by the argument "string".

Examples: If INPUT had the value

```
'E123456.XYZ.DATA'
```

the statement

```
% DATASET = DSN(INPUT);
```

would assign the value

```
E123456.XYZ.DATA
```

to the variable DATASET.

If INPUT had the value

```
XYZ.DATA
```

and the current user was E654321 then the statement above would cause the value

```
E654321.XYZ.DATA
```

to be assigned to DATASET.

A.13.2 SUBSTR Built-in Function

Definition: SUBSTR extracts a substring of programmer defined length from a given string and returns the substring to the point of invocation.

Reference: SUBSTR (string,[i ,j])

Arguments: The argument "string" represents the string from which a substring will be extracted. Argument "i" represents the starting point of the substring and "j" represents the length of the substring.

Arguments "i" and "j" must be integers or variables with integer values. Assuming that the length of "string" is k, arguments "i" and "j" must satisfy the following conditions:

(1) j must be less than or equal to k and greater than or equal to 0.

(2) i must be less than or equal to k and greater than or equal to 1.

(3) The value of i + j - 1 must be less than or equal to k.

Thus, the substring, as specified by "i" and "j" must lie within "string". If "j" is not specified, it is assumed to be equal to the value of k - i + 1. In other words, it is assumed to be the length of the remainder of "string", beginning at the ith position in "string".

Result: The value returned by this function is a character string determined as follows:

(1) If j=0, the returned value is the null string.

(2) If j is greater than 0, the returned value is that substring beginning at the ith character of the first argument and extending j characters.

(3) If j is not specified, the returned value is that substring beginning at the ith character and extending to the end of "string".

Example: If CHARGE is the character string

```
21-3715 5407
```

the statement

```
WO = SUBSTR(CHARGE,4,4);
```

will cause a 4-character substring to be extracted from CHARGE, starting at the 4th character. This substring,

```
3715
```

will then be assigned to the variable WO.

A.14 MAKING A PSI MACRO AVAILABLE FOR USE

Once a PSI macro has been written it must be copied into the PSI macro library before it can be used with RUNPROG. This library is a partitioned data set where each member is a macro. The member name should be exactly the same as the name given the macro.

A special DCAS command is available for putting a new member in the library or changing an existing member. This command is named ADDMAC and has two positional parameters. The first is the name of a control (CNTL) data set containing the macro to be added or replaced and the second is the name of the macro. For example,

```
ADDMAC  XYZ  P1234
```

will put the data set userid.XYZ.CNTL into the macro library with member name P1234. Notice that the data set specified must be of type CNTL and exist in the library of the current user and neither the userid or type are specified on the ADDMAC command.

An existing macro can be changed by first copying the appropriate member of the macro library (which is named TSOGURU.DCASJCL.CNTL), making the desired changes in the copy and using ADDMAC with this new data set and the original member name.

A.15 REMOVING A PSI MACRO FROM THE LIBRARY

A PSI macro can be deleted from the library by use of the DELMAC command. This command is written as follows:

```
DELMAC macro-name
```

where macro-name is the name of the macro to be removed. DELMAC will give the user the opportunity to specify additional macros to be deleted by prompting for more names, thus, allowing several macros to be deleted at once.

A.16 EXERCISING A PSI MACRO

The DCAS command, RUNPROG, is used to invoke a PSI macro to build and submit a job. Its form is as follows:

RUNPROG macro-name [keyword(value)] ...

Here, 'macro-name' is the name of the PSI macro to be used and '[keyword(value)] ...' is a (possibly empty) list of keyword parameters and values as needed for the particular run. The following ten keyword parameters may be used with RUNPROG.

(1) SOURCE
(2) DATA
(3) OBJECT
(4) CASEDATA
(5) MATDATA
(6) MADOL

\- data-set names

(7) XCHAR - character(s) to be appended to userid (or jobid) for use as a jobname
(8) MDPROG - program name
(9) MOREPARM - indication as to whether more parameter values are to be specified by the user
(10) SUBMIT - indication as to whether the built job is to be submitted.

The first eight of these are provided merely as a convenience; they may be omitted here even if it is desired to override their default values (see Section A.7). Values supplied for the data-set names (parameters 1-6) can be specified according to TSO conventions. RUNPROG will automatically add the userid if necessary (that is, RUNPROG will perform the DSN function (see Section A.13) on supplied data-set names). The seventh parameter allows the specification of a character or string of characters to be used in the construction of a jobname for the job to be

submitted. The jobname will be the userid or jobid, if one has been established for the current user, followed by the character(s) of XCHAR. If the jobname so formed is greater than eight characters, it will be truncated to eight. The eight parameter is merely some name. It can be given any value, up to eight characters in length.

The ninth parameter is used to indicate that additional parameter values (other than those given on the RUNPROG statement) are to be provided by the user. This can be done by specifying MOREPARM(YES); the default value is NO (see Section A.8). The tenth parameter indicates whether or not the job being built is to be submitted for batch execution. The default value is YES. If SUBMIT(NO) is specified then RUNPROG will build and list a complete job but will not submit it. It will save the job in a data-set named JOB.CNTL in the current user's library. If SUBMIT(LIST) is specified, the job will be listed and saved as JOB.CNTL as well as being submitted.

It should be noted that RUNPROG is a TSO command procedure (CLIST) and as such allows the use of abbreviations for keyword parameter names. Only enough characters need be specified to insure the unique identification of a given parameter. For example, one could write,

```
RUNPROG P1234 SO(SOURCE.FORT) D(DATA.DATA) O(OBJECT.DATA) X(A)
       MO(YES) SU(NO)
```

A.17 TESTING A PSI MACRO

A PSI macro can (and should) be tested with the TESTMAC command before placing it into the macro library. This command is very similar to the RUNPROG command except for the following:

(1) It accesses a macro existing as a data set in the library of the individual using the TESTMAC command.

(2) The Job Control Language (JCL) statements produced by processing the macro will not be submitted but will be displayed at the terminal as they are being built. Prompting messages, if any, will appear as they are encountered within the macro. Hence they may be interspersed with the JCL statements being built. These messages should be answered as usual.

The form of the TESTMAC command is as follows:

TESTMAC data-set-name [keyword(value)] ...

The 'data-set-name' specifies the name of the data set containing the macro to be tested. The name must include the type qualifier. The keyword parameters are specified in exactly the same manner as for the RUNPROG command. All but the SUBMIT keyword can be used with TESTMAC. The following is an example demonstrating the use of TESTMAC to check a newly written macro, created as a TSO data set named P2941.CNTL, where a value for the keyword DATA and an indication that more parameters are to be supplied are given:

```
TESTMAC P2941.CNTL DATA(P2941.DATA) MOREPARM(YES)
```

A.18 SPECIFYING DATA-SET NAMES TO RUNPROG

In TSO a user may refer to a catalogued data-set outside his own library by enclosing the name in quotes. Unfortunately quotes are also used as delimiters in TSO and PL/I (the language of RUNPROG). This requires that one often write two successive quotes to indicate one actual quote character. Several levels of passing these characters can lead to a proliferation of quotes.

Without going into the reasons why, the following rules are given as a guide for specifying a quoted data-set name to RUNPROG.

(1) When specifying a data-set as SOURCE, OBJECT, DATA, CASEDATA, MATDATA, or MADOL as part of the RUNPROG statement or when specifying such a data-set with any of the DCAS commands, PRNT, PNCH, PRNTOUT, PRNTPDS, PRNTR, PNCHR, PRNTOUTR, GETDS or PUTDS, the name must be preceded by four (4) quotes and followed by four (4) quotes.

(2) When the data-set is specified as an additional parameter (using the MOREPARM(YES) option), the name must be both preceded and followed by three (3) quotes.

(3) When prompted for a data-set name, only single quotes are required both preceding and following it.

Data-sets which are within the user's library are specified identically in all cases; that is, the complete data-set name, exclusive of the 'userid' and its following decimal point is given.

A.19 EXAMPLE OF A PSI MACRO

The following Figure is an example of a PSI macro. Card 1 is just a comment. Card 2 is a macro definition statement giving the macro the name, EXAMPLE, and establishing default values for the parameters, TIME, PRIORITY, PROGRAM#, and CLASS. Card 3 tests the variable, TIME. If TIME is not a 2 (i.e. a value for TIME was specified using the MOREPARM(YES) option) the statement on card 4 is executed. Hence PRIORITY will be 3 if TIME is 2 and 0 otherwise. Cards 5 through 8 are JCL cards in which parameter substitutions will be made. JOBNAME, $USERID, USERNAME, $CSR#, $DP#, $GP#, and BIN# are automatically determined based on the logon information. The other parameters get their values from the defaults specified on card 2, from information given by the user with the MOREPARM(YES) option, or as determined by the macro itself (as possibly for PRIORITY). Card 9 is another comment. Card 10 tests the variable, PLOTS. Since PLOTS

```
C                    EXAMPLE OF A PSI MACRO                          00000001
% EXAMPLE MACRO  TIME=2, PRIORITY=3, PROGRAM#=1234, CLASS='B' ;       00000002
%     IF TIME '= 2 THEN                                              00000003
%         PRIORITY = 0 ;                                             00000004
//<JOBNAME> JOB (PROGRAM#),<&USERID>,PRTY=(PRIORITY),CLASS=(CLASS),   00000005
// REGION=70K,NOTIFY=(&USERID)                                       00000006
/*ACC(USERNAME) <@19&CSR#> <@33'1'> <@34PROGRAM#> <@44DF#> <@4846F#>  *00000007
/*<@5'BIN#>                                                          00000008
C                    TEST FOR PLOTS                                  00000009
%     IF PLOTS = 'YES' THEN                                          00000010
%         DO ;                                                       00000011
/*OPERATOR    JOB NAME=<JOBNAME>, PRTY=(&PRTY), PROGRAMMER=(USERNAME), 00000012
/*OPERATOR    BIN=<BIN#>, PLOTTER=CALCOMP, NO. FILES=1, DEPT=<4DF#>,  00000013
/*OPERATOR    SECTION=<&6F#>, CSR=<&CSR#>, PLOT PAPER=(PAPER)         00000014
/*SETUP  DDNAME=PLOTTAPE,DEVICE=2400-9,ID=(,SAVE,NL)                 00000015
%         END ;                                                      00000016
//      EXEC  FORTRAN                                                00000017
//C.SYSIN  DD  DSNAME=<SOURCE>,DISP=SHR                              00000018
//L.OBJECT DD  DSNAME=<OBJECT>,DISP=SHR                              00000019
//L.SYSIN  DD  *                                                     00000020
  INCLUDE COMPILE                                                   00000021
  INCLUDE OBJECT                                                    00000022
  ENTRY MAIN                                                        00000023
C                    TEST FOR OUTPUT DATA-SET                        00000024
%     IF OUTPUT '= 'NO' THEN                                         00000025
%         DO ;                                                       00000026
%             OUTPUT = DSN'(OUTPUT) ;                                00000027
//G.FT07F001  DD  DSNAME=<OUTPUT>,UNIT=TSO,DISP=(NEW,CATLG),         00000028
// SPACE=(TRK,(2,10)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680)           00000029
%         END ;                                                      00000030
%     IF PLOTS = 'YES' THEN                                          00000031
//G.PLOTTAPE  DD  UNIT=2400,LABEL=(,BLP)                             00000032
%         IF PLOTS '= 'YES' THEN                                     00000033
//G.PLOTTAPE  DD  DUMMY                                              00000034
//G.FT05F001  DD  DSNAME=<DATA>,DISP=SHR                             00000035
//                                                                   00000036
%     END EXAMPLE ;                                                  00000037
```

Figure. Example of a PSI Macro

is not a system variable, and no default has been established for it, the user will be prompted for a value (unless he has already supplied one using the MOREPARM(YES) option). If the user supplied value is 'YES', that is, there will be plots produced, cards 12 through 15 (those between the DO statement on card 11 and the END statement on card 16) will be included in the job being built. Otherwise, they will not. Thus, appropriate operator instructions and a SETUP card will be included if there is to be plot output. In this case, the values for JOBNAME, PRIORITY, USERNAME, BIN#, $DP#, $GP#, and $CSR# will be substituted as before. The user will be prompted for a value for the variable, PAPER, unless he gave one earlier, and that value will be substituted in card 14. Cards 18 through 23 are JCL statements. The values, if any, supplied by the user on the RUNPROG statement for SOURCE and OBJECT will be substituted in card 18 and 19. If no values were given the system defaults will be used. Card 24 is again a comment. Card 25 is another test, checking to see if an output data is to be saved. The user will be prompted for a value for OUTPUT unless he previously gave one, and the statements between the DO on card 26 and the END on card 30 will be executed only if the value of OUTPUT is not 'NO'. In that case its value will be taken as the name to be given the output data set. The statement on card 27 performs the DSN function on the name (see Section A.13), converting it from a valid TSO specification of a data set name in an OS acceptable form. The substitution of this name is then made to the DD statement given on cards 28 and 29. Cards 31 and 33 check the value of PLOTS again and the appropriate DD card, either 32 or 34 will be included. Card 35 and 36 are JCL cards. A value will be substituted for the variable DATA in card 35 as was specified by the user or by default if none was specified. Finally card 37 indicates the end of the macro.

APPENDIX B

THE INPUT DESCRIPTION LANGUAGE

B.1 INTRODUCTION

The ID (Input Description) language is just an extension of the PSI (Program Setup Instructions) language. The basic rules for constructing ID macros are the same as for PSI macros, and all statements which are legal in PSI macros are legal in ID macros. The additional language constructs for input description are given in the following sections.

B.2 DESCRIBING INPUT PARAMETERS

A character string giving the meaning of an input parameter (or any macro variable) can be associated with its name by means of a DEFINE statement. The form of this statement is as follows:

```
% DEFINE identifier (character string);
where 'identifier' is any variable name
      'character string' is any string of characters
```

For example, the string 'MACH NUMBER' can be associated with the variable name 'M' as follows:

```
% DEFINE M(MACH NUMBER);
```

A more involved description is also possible as indicated by the following example.

```
% DEFINE PRNT(PRINT OPTION -
                PRNT=0 FOR STANDARD PRINT,
                PRNT=1 FOR SUMMARY PRINT ONLY);
```

Here the variable, PRNT, is a flag by which the user may specify one of two output options.

Once a parameter has been so defined a user may obtain the associated description by typing a '?' followed by the parameter name. This may be done whenever a response of some kind is expected from the user. This action interrupts the normal processing and, after providing the information requested, execution of the macro resumes automatically.

B.3 DESCRIBING INPUT FORMATS

In order to describe the formats in which a program expects its input to be given the PUT statement has been expanded to allow a format specification. Syntactically this statement resembles the PUT EDIT statement in PL/I. It is written as follows:

```
% PUT (identifier [,identifier] ...) (format list)
where 'identifier' is any variable name
      'format list' is any valid PL/I format list
```

Upon execution of this statement the specified list is written on the output file according to the format specification. An example of this form of the PUT statement is given below.

```
% PUT (X,Y,Z) (A(3),X(2),2A(5));
```

B.4 SPECIFYING PARAMETER TYPES

Parameters may be declared to be of one of three types - character, integer, or real. This is accomplished with the TYPE state-

ment. It is written as follows:

% TYPE (identifier [,identifier] ...) { CHAR / INTEGER / REAL }

[,(identifier [,identifier] ...) { CHAR / INTEGER / REAL }] ... ;

where 'identifier' is any variable name

For example, one could specify X and Y to be real numbers; I, J, and K to be integers; and ABC to be a character string as follows:

% TYPE (X,Y) REAL,(I,J,K) INTEGER, (ABC) CHAR;

When a parameter has been so typed, any value supplied for it will be checked for compliance. If the value is not of the proper type, the user is immediately notified and prompted for a new value if the macro is being executed interactively. Otherwise, an error message is written on the output file.

B.5 SPECIFYING PARAMETER LIMITS

A range or set of acceptable values can be specified for a parameter with the LIMIT statement. It can be written as follows:

% LIMIT range [,range] ... ;

where 'range' can be one of the following:

identifier = (constant [,constant]...)

identifier { < / <= / = / > / >= } constant

$$\text{constant} \left\{ \begin{matrix} < \\ <= \end{matrix} \right\} \text{identifier} \left[\left\{ \begin{matrix} < \\ <= \end{matrix} \right\} \text{constant} \right]$$

$$\text{constant} \left\{ \begin{matrix} > \\ >= \end{matrix} \right\} \text{identifier} \left[\left\{ \begin{matrix} > \\ >= \end{matrix} \right\} \text{constant} \right]$$

where 'identifier' is any variable name
'constant' is any valid constant

With the LIMIT statement a parameter can be required to be one out of a set of values or to lie between two values. Multiple ranges or sets can be given for a single parameter in which case they are logically OR-ed together. For example, to specify that the variable X must be either between 0 and 1 or the value -99, one could write,

```
% LIMIT 0 <= X<= 1, X = -99;
```

Several parameters may be given limits with the same statement as in the following example:

```
% LIMIT 0 <= I < = 10, X > 0, A = ('K','L')
```

This statement says that I must lie between 0 and 10 (inclusive), that X must be greater than 0 and that A must have either 'K' or 'L' as a value.

APPENDIX C

"ASSIST" SURVEY RESULTS

C.1 INTRODUCTION

On May 20, 1976 a questionnaire was distributed to all DCAS users outside the Administrative/Computer Services organization. The purposes of the questionnaire were to evaluate the Program Submittal (RUNPROG) component of ASSIST (A Scientific Software Interface System for Terminal-users) and to obtain information for guiding the development of additional capabilities. The results of 28 completed questionnaires are presented and analyzed in the following section.

C.2 RESULTS

The first question attempted to get an indication of the distribution of users according to their degree of activity on the system. The question and number giving each response are given in Table 1. The results indicate a fairly even distribution.

Question two tried to determine whether users felt the RUNPROG capability to be of help in their work. The results were overwhelmingly affirmative. The primary reasons sighted for this centered around the fact that RUNPROG gave them a simple, convenient means for getting a job submitted and run quickly. The complete responses to this question are given in Table 2.

The third question asked users to describe any difficulties they had experienced with RUNPROG. The major problems stated were that the RUNPROG command itself was slow to execute and that the documentation for it was incomplete or confusing. Table 3 contains the complete set of responses.

Please estimate your activity on DCAS as follows:

a) Terminal use

7 (25%) heavy user (more than 10 hours/week)
7 (25%) moderate user (5 - 10 hours/week)
5 (18%) light user (1 - 5 hours/week)
9 (32%) occasional user (less than 1 hour/week)

b) RUNPROG use

4 (14%) heavy user (more than 20 times/week)
9 (32%) moderate user (10 - 20 times/week)
5 (18%) light user (3 - 10 times/week)
10 (36%) occasional user (less than 3 times/week)

Table 1. DCAS Activity

Do you feel that the RUNPROG capability is a benefit in your present work?

24 (86%)	yes
1 (4%)	no
1 (4%)	yes & no
2 (7%)	no response

If so how?

6 (21%)	it improves job turnaround time
5 (18%)	it provides faster way to submit a job
5 (18%)	it's simplier than generating JCL
3 (11%)	it's essential
2 (7%)	it allows direct submittal by user
2 (7%)	it's convenient
1 (4%)	it's efficient
1 (4%)	it reduces errors

If not, why not?

1 (4%)	it's too slow
1 (4%)	it's not applicable

Table 2. Benefit of RUNPROG

What difficulties (if any) have you experienced in your use of RUNPROG?

6 (21%)	it's too slow
6 (21%)	the options are confusing (poor documentation and non-standard keywords)
2 (7%)	machine problems
1 (4%)	lost output
1 (4%)	hard to run multiple cases
1 (4%)	many user errors not caught (e.g. incorrect data set name)

Table 3. RUNPROG Difficulties

Question four asked users what changes were felt to be necessary in RUNPROG. The most common response was that they wanted it to run faster. The results of this question are given in Table 4.

In question five users were asked to rate RUNPROG by comparing it to past methods of operation. Approximately 43% felt it was a significant improvement and 64% felt it was at least some improvement. Unfortunately 32% did not respond at all, most of them indicating they did not know what they were supposed to compare RUNPROG to. The intent of the question was to determine whether users felt that having a capability to submit jobs from a terminal, without having to be concerned with JCL, was better than giving instructions to a programmer and having him submit the job. Even though many seemed to misunderstand the question, the responses from those who did clearly indicate a great satisfaction and acceptance of RUNPROG. The complete responses to this question are given in Table 5.

Question six asked users to express their opinions regarding the usefulness of a capability to obtain information describing available programs from the terminal. A majority (54%) of the users strongly favored the addition of such a capability, and 32% thought it might be useful. Most thought it was important to have access to such information, but many felt on-line access was not necessary. The responses are listed in Table 6.

Question seven asked users for their opinions of an interactive capability for assisting in preparing program input. Again 54% felt such a capability would definitely be beneficial. Another 21% said "maybe" while 18% said "no" and 7% did not respond. Those favoring the addition of this capability thought it would reduce errors and make input preparation easier. Those responding negatively generally could not see how it would help them in their work. Table 7 gives the complete set of responses.

What changes (if any) would you like to see in the way RUNPROG operates?

6 (21%)	improved response
2 (7%)	more flexibility in overriding JCL
2 (7%)	check data set names for correctness
1 (4%)	be able to view JCL and correct mistakes before submittal
1 (4%)	better terminal availability
1 (4%)	be able to enter multiple cases at once
1 (4%)	be able to reuse a data set without making a copy

Table 4. RUNPROG Improvements

What is your overall rating of the RUNPROG capability?

12 (43%)	a significant improvement over past methods of operation
6 (21%)	somewhat better than past methods
1 (4%)	comparable to past ways of operating
0 (0%)	somewhat worse than past methods
0 (0%)	a definite step backwards
9 (32%)	no response

Table 5. RUNPROG Rating

Currently being designed is an on-line capability which would allow DCAS users to obtain information describing available computer programs (i.e., a capability enabling remote users to get a list of program titles and optionally abstracts for existing software in a specified category). Would you find such a capability useful?

15 (54%)	yes
6 (14%)	no
9 (32%)	maybe

Explain:

8 (29%)	it would make information conveniently available
6 (21%)	it's not necessary to have an on-line capability; a batch capability would be sufficient
3 (11%)	it would reduce duplication of software development
2 (7%)	it should not impact overall DCAS response
1 (4%)	it would increase use of existing programs
1 (4%)	information must be kept up-to-date
1 (4%)	fellow workers are a better source of information

Table 6. Information Retrieval Capability

Also in the design stage is an interactive capability which would assist DCAS users in preparing input data for program selected to run (i.e., a capability which would prompt users for information and properly format it to meet program input specifications). Would you find such a capability useful?

15 (54%)	yes
5 (18%)	no
6 (21%)	maybe
2 (7%)	no response

Explain:

5 (18%)	it should not slow down overall response
4 (14%)	it would reduce input errors
3 (11%)	it would help the novice or occasional user
2 (7%)	it would be more convenient
2 (7%)	EDIT is usually sufficient for input preparation
2 (7%)	no foreseen use
1 (4%)	it would be more efficient
1 (4%)	it would save time
1 (4%)	it would be helpful for FAMAS matrices
1 (4%)	it should provide user with enough information
1 (4%)	it must be optional
1 (4%)	it would be impractical for large amounts of data

Table 7. Input Preparation Capability

Question eight asked users what other capabilities they would like to see incorporated into DCAS. Most users wanted capabilities available under TSO but not included in the TSO subset provided with DCAS. Users also indicated they wanted better response. The complete list of responses is given in Table 8.

Finally, question nine gave users a chance to make any other comments regarding DCAS. The need for better response was again mentioned, and a desire to have DCAS available for a longer period was also expressed. All the comments given are listed in Table 9.

C.3 CONCLUSIONS

Based upon the responses to the questionnaire, the following conclusions can be drawn:

(1) The RUNPROG capability is a highly accepted and useful tool for DCAS users and, therefore, should be fully developed.

(2) Poor response is the biggest problem with RUNPROG and DCAS in general. Efforts to make improvements in this area must be initiated.

(3) The documentation of RUNPROG and other DCAS capabilities is inadequate and should be improved.

(4) The development of a capability to obtain information about existing programs from a terminal would probably be beneficial. It might be sufficient, however, to allow only the request for information to be made at the terminal and have the actual information found and printed in a batch mode.

(5) The development of a capability to assist in the preparation and checking of program input data would help many DCAS users, but the use of this capability should be optional

What other capabilities would you like to see incorporated into DCAS that would make it easier for you to use existing applications software?

2 (7%)	better response
2 (7%)	foreground execution capability
2 (7%)	improved documentation
1 (4%)	expanded CLIST capability
1 (4%)	all of the TSO commands
1 (4%)	ability to print data sets with column and title headings
1 (4%)	reduced default track allocations for data sets
1 (4%)	more user disk space
1 (4%)	capability to store data at terminals on paper tape or cassettes

Table 8. DCAS Improvements

Additional Comments:

4 (14%)	need better response
2 (7%)	need foreground execution capability
2 (7%)	need longer DCAS availability (6AM to 6PM)
1 (4%)	need better machine reliability
1 (4%)	need capability to list contents of a PANVALET library member
1 (4%)	need capability to get a list when punching a data set
1 (4%)	should be able to specify that punch be interpreted
1 (4%)	the print of a data set should start at the top of a page
1 (4%)	log-on time should be reduced
1 (4%)	users should be taught to read JCL
1 (4%)	it's a useful system

Table 9. Comments

REFERENCES

1. Computer Graphics Designer's Manual, 1TMC-1, Lockheed-California Company, Burbank, California, 1970.

2. Conversational Programming System (CPS), Terminal User's Manual, GH20-0758, IBM Corporation, White Plains, New York, 1970.

3. "Flutter And Matrix Algebra System (FAMAS) Manual," LR 23657, Lockheed-California Company, Burbank, California, 1970.

4. IBM System/360 Operating System: Time Sharing Option Guide, GC28-6698, IBM Corporation, White Plains, New York, 1973.

5. Anderson, R. H. and Sibley, W. L., "A New Approach to Programming Man-Machine Interfaces," R-876-ARPA, The Rand Corporation, Santa Monica, California, 1972.

6. Ashok, D., "User Requirements in Man-Machine Interactive Systems," SWRL-PP-20, Southwest Regional Educational Laboratory, Inglewood, California, 1972.

7. Carlisle, J. H., "Interactive Man-Machine Communication," TR-51, Yale University, New Haven, Connecticut, 1972.

8. Elson, M., Concepts of Programming Languages, Science Research Associates, Inc., Chicago, 1973.

9. Fajman, R. and Borgelt, J., "Wylbur: An Interactive Text Editing and Remote Job Entry System," Comm. ACM 16, 5 (May 1973), 314-322.

10. Garrocq, C. A. and Hurley, M. J., "The IPAD System: A Future Management/Engineering/Design Environment," Proc, ACM/IEEE 11th Design Automation Workshop, Denver, Colorado, June 1974.

11. Gries, D., Compiler Construction for Digital Computers, Wiley, New York, 1971.

12. Lingard, R. W., "Engineer Oriented Remote Computing," LR 27518, Lockheed-California Company, Burbank, California, 1975.

13. Marks, S. L., "The JOSS Years: Reflections on an Experiment," R-918, The Rand Corporation, Santa Monica, California, 1971.

14. Martin, J., Design of Man-Computer Dialogues, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1973.

15. Notestine, R. E., "Graphics and Computer Aided Design in Aerospace," Proc. AFIPS 1973 NCC, Vol. 42, AFIPS Press, Montvale, New Jersey.

16. Sackman, H., "Rudiments of a Real-World Theory of Man-Computer Problem Solving," R-1491-NSF, The Rand Corporation, Santa Monica, California, April 1974.

17. Shaw, J. C., "JOSS: A Designer's View of an Experimental On-Line Computing System," P-2922, The Rand Corporation, Santa Monica, California, 1964.

18. Sproul, R. G., "Parametric Surfaces in a Computer Graphics Design System," LR 26863, Lockheed-California Company, Burbank, California, 1974.

19. Thompson, J. M., "The DISKEDIT System - On-Line Data Management," LR 26797, Lockheed-California Company, Burbank, California, 1974.

20. Wasserman, A. I., "The Design of 'Idiot-Proof' Interactive Programs," Proc. AFIPS 1973 NCC, Vol. 42, AFIPS Press, Montvale, New Jersey.

21. Wilczynski, D., "An Approach to Run Time Modification of Data With Special Application for MADOL Programs," LR 23529, Lockheed-California Company, Burbank, California, 1970.

ENGINEERING REPORT INITIAL DISTRIBUTION LIST

(SEE EPM 4-07)

REPORT NO. LR 28005

PAGE 1 OF 3

TITLE	MODEL	SECURITY CLASS.	DATE
SUMMARY OF 1976 INDEPENDENT RESEARCH ON ENGINEER ORIENTED REMOTE COMPUTING	I.R.	UNCLASSIFIED	7-28-77

APPROVALS

DIVISION ENGINEER: R B Berry

ORIGINATING ORGANIZATION (TITLE & DEPT. NO.)
Scientific Analytical Programming (80-36)
Scientific Computing Division

COMMERCIAL ENGINEERING (COMMERCIAL ENGINEERING BRANCH REPORTS)
Don McNeely

WO/EWA 21 (CLASS) 3715 (WORK ORDER) 5407 (E.W.A.)

PRODUCT EVALUATION GROUP
Dimon for L. C. Turner

REMARKS: NONE

LEGAL BRANCH - PATENT SECTION (STATE ANY RESTRICTIONS)
Public Domain

LIMITATION ON ACCESS TO DATA:

UNLESS LIMITATIONS ON SUBSEQUENT RELEASE OF THIS REPORT ARE STATED BELOW, COPIES WILL BE MADE FREELY ACCESSIBLE TO ALL CORPORATION EMPLOYEES. (IF LIMITED, SUBSEQUENT RELEASE TO OTHER ORGANIZATIONS REQUIRES COMPLETION OF FORM 7229.)

LIMITED TO: _____

REASON: _____

DATE ON WHICH LIMITATION MAY BE LIFTED: _____

WOULD IT BE BENEFICIAL TO CALAC TO RELEASE THIS REPORT TO THE PUBLIC VIA NASA/DoD LIBRARIES? ☒ YES ☐ NO
(ANSWER THIS QUESTION FOR INDEPENDENT RESEARCH OR INDEPENDENT DEVELOPMENT FUNDED REPORTS ONLY.)

COPY NO.	DISTRIBUTION 1. ASSIGN COPY NO. TO HARD COPIES ONLY. 2. LIST MICROFICHE AND ABSTRACT RECIPIENTS LAST. 3. EXTERNAL COPIES: INDICATE TRANSMITTER. 4. CIRCLE COPY NO. OF REPORTS ALREADY DISTRIBUTED.	PUT "X" IN PROPER COLUMNS: EXTERNAL (NON-LAC)	INTERNAL (WITHIN LAC)	TYPE: HARD COPY	MICRO FICHE	ABSTRACT ONLY	LIBRARY CIRCULATION	NO REVISIONS
MASTER	INDICATE WHERE FILED: ☐ REPORTS SERVICES GROUP ☐ PUBLICATION SERVICES GROUP, _____ PROJECT		x	x				
1	VITAL RECORDS (TO BE TRANSMITTED BY REPORTS SERVICES GROUP)		x	x				
2	REPORTS SERVICES GROUP		x	x				
3, 4	CENTRAL LIBRARY		x	x				
5	A. N. Baker 70-01 63 A-1		X	X				
6	D. L. Bickel 86-11 67 A-1		X	X				
7	B. L. Bivens 80-36 67 A-1		X	X				
8	P. Brunelli 28-06 146 B-1		X	X				
9	A. L. Byrnes 75-41 63 A-1		X	X				
10	L. C. Cowgill 75-72 63 A-1		X	X				
11	D. M. Crawford 86-11 67 A-1		X	X				
12	D. R. Crawford 80-37 67 A-1		X	X				
13	A. R. Curtis 72-71 311 B-6		X	X				
14	R. D. Elliott 75-41 63 A-1		X	X				
15	M. T. Evans 86-11 67 A-1		X	X				
16	R. Harris, Jr. (GELAC) 87-14 B-1B 274		X	X				

CALAC FORM 5759-5

ENGINEERING REPORT INITIAL DISTRIBUTION LIST

(SEE EPM 4-07)

REPORT NO. LR 28005

TITLE	MODEL	SECURITY CLASS.	DATE
	I. R.	UNCLASSIFIED	7-28-77

APPROVALS

ORIGINATING ORGANIZATION (TITLE & DEPT. NO.)

DIVISION ENGINEER

COMMERCIAL ENGINEERING
(COMMERCIAL ENGINEERING BRANCH REPORTS)

WO/EWA 21 (CLASS) 3715 (WORK ORDER) 5407 (E.W.A.)

REMARKS

PRODUCT EVALUATION GROUP

LEGAL BRANCH - PATENT SECTION (STATE ANY RESTRICTIONS)

LIMITATION ON ACCESS TO DATA:

UNLESS LIMITATIONS ON SUBSEQUENT RELEASE OF THIS REPORT ARE STATED BELOW, COPIES WILL BE MADE FREELY ACCESSIBLE TO ALL CORPORATION EMPLOYEES. (IF LIMITED, SUBSEQUENT RELEASE TO OTHER ORGANIZATIONS REQUIRES COMPLETION OF FORM 7229.)

LIMITED TO: _____

REASON: _____

DATE ON WHICH LIMITATION MAY BE LIFTED: _____

WOULD IT BE BENEFICIAL TO CALAC TO RELEASE THIS REPORT TO THE PUBLIC VIA NASA/DoD LIBRARIES? ☐YES ☐NO
(ANSWER THIS QUESTION FOR INDEPENDENT RESEARCH OR INDEPENDENT DEVELOPMENT FUNDED REPORTS ONLY.)

COPY NO.	DISTRIBUTION 1. ASSIGN COPY NO. TO HARD COPIES ONLY. 2. LIST MICROFICHE AND ABSTRACT RECIPIENTS LAST. 3. EXTERNAL COPIES: INDICATE TRANSMITTER. 4. CIRCLE COPY NO. OF REPORTS ALREADY DISTRIBUTED.	PUT "X" IN PROPER COLUMNS							
		EXTERNAL (NON-LAC)	INTERNAL (WITHIN LAC)	TYPE: HARD COPY	TYPE: MICRO FICHE	TYPE: ABSTRACT ONLY	TYPE: LIBRARY CIRCULATION	NO REVISIONS	
MASTER	INDICATE WHERE FILED: ☐REPORTS SERVICES GROUP ☐PUBLICATION SERVICES GROUP, _____PROJECT		x	x					
17	F. W. Johnson 75-73 90 A-1		x	x					
18	T. R. Jones 80-36 67 A-1		x	x					
19	D. K. Kawamoto 80-34 67 A-1		x	x					
20	P. H. Kretsinger 80-36 67 A-1		X	X					
21	R. W. Lingard 80-36 67 A-1		X	X					
22	J. D. Little 80-36 67 A-1		X	X					
23	J. J. Lucas 80-34 67 A-1		X	X					
24	R. F. O'Connell 75-71 63 A-1		X	X					
25	R. B. Ostrom 75-72 63 A-1		X	X					
26	R. R. Plank 70-01 63 A-1		X	X					
27	W. L. Rakness 75-72 90 A-1		X	X					
28	S. W. Robinson 80-36 67 A-1		X	X					
29	D. H. Saiki 80-36 67 A-1		X	X					
30	J. E. Sherman (LMSC) 19-40 102		X	X					
31	G. E. Smith 80-01 67 A-1		X	X					

ENGINEERING REPORT INITIAL DISTRIBUTION LIST

(SEE EPM 4.07)

REPORT NO. IR 28005

TITLE

MODEL: I.R.

SECURITY CLASS.: UNCLASSIFIED

DATE: 7-28-77

APPROVALS

DIVISION ENGINEER

ORIGINATING ORGANIZATION (TITLE & DEPT. NO.)

COMMERCIAL ENGINEERING (COMMERCIAL ENGINEERING BRANCH REPORTS)

WO/EWA 21 (CLASS) 3715 (WORK ORDER) 5407 (E.W.A.)

REMARKS

PRODUCT EVALUATION GROUP

LEGAL BRANCH - PATENT SECTION (STATE ANY RESTRICTIONS)

LIMITATION ON ACCESS TO DATA:

UNLESS LIMITATIONS ON SUBSEQUENT RELEASE OF THIS REPORT ARE STATED BELOW, COPIES WILL BE MADE FREELY ACCESSIBLE TO ALL CORPORATION EMPLOYEES. (IF LIMITED, SUBSEQUENT RELEASE TO OTHER ORGANIZATIONS REQUIRES COMPLETION OF FORM 7229.)

LIMITED TO:

REASON:

DATE ON WHICH LIMITATION MAY BE LIFTED:

WOULD IT BE BENEFICIAL TO CALAC TO RELEASE THIS REPORT TO THE PUBLIC VIA NASA DoD LIBRARIES? ☐YES ☐NO
(ANSWER THIS QUESTION FOR INDEPENDENT RESEARCH OR INDEPENDENT DEVELOPMENT FUNDED REPORTS ONLY.)

COPY NO.	DISTRIBUTION 1. ASSIGN COPY NO. TO HARD COPIES ONLY. 2. LIST MICROFICHE AND ABSTRACT RECIPIENTS LAST. 3. EXTERNAL COPIES: INDICATE TRANSMITTER. 4. CIRCLE COPY NO. OF REPORTS ALREADY DISTRIBUTED.				PUT "X" IN PROPER COLUMNS: EXTERNAL (NON-LAC)	INTERNAL (WITH LAC)	TYPE: HARD COPY	MICRO FICHE	ABSTRACT ONLY	TITLE PAGE ONLY	NO REVISION
MASTER	INDICATE WHERE FILED: ☐REPORTS SERVICES GROUP ☐PUBLICATION SERVICES GROUP, _____ PROJECT					x	x				
32	J. F. Stroud	75-42	63	A-1		x	x				
33	H. P. Weinberger	80-36	67	A-1		x	x				
34	H. H. Hara	83-01	67	A-1		x	x				
35	NASA Library										
36	DDC										